III. Remarks

Applicant is grateful to Examiner Nguyen for her time and consideration extended to the undersigned and the inventor in the telephone interview of April 28, 2005. At the conclusion of the interview, the Examiner stated that Applicant did not have to file an interview summary and that one would be forthcoming.

Claims 1, 9, 17, and 20 have been amended as described in more detail below. Claims 3, 11, 19 and 22 have been amended to correct formalistic errors in the claim language.

Reconsideration and withdrawal of the rejections are respectfully requested in view of the following arguments and foregoing amendments.

A. Rejection under 35 U.S.C. § 102

The Action rejects Claims 1, 5, 8, 9, 16, 17 and 20 as being anticipated by U.S. Patent No. 5,867,382 to McLaughlin.

As discussed in the interview, Applicant's claimed invention is directed to a method of programming a programmable logic controller (PLC) to direct a process, e.g., an assembly process. The Applicant has realized that there are difficulties in debugging prior art programs, such as ladder logic programs, and in reprogramming these programs to perform different sequences of operations. Indeed, skilled programmers are required for these programming tasks. Based on this realization, the Applicant developed an intuitive graphical user interface that allows a person without any knowledge of any particular program language to program and to the PLC control program and to debug it.

As discussed in the interview, the claimed method is best understood with reference to FIGS. 2 and 3 of the application. FIG. 2 shows a graphical user interface that allows a programmer to select inputs and outputs for the PLC for a series of sequential steps to be preformed in a process, such as an assembly process. The selections are converted to an input control table and an output control data table that can then be provided to a PLC, which performs

the sequential steps in the process according to the programmed data in the control tables. This methodology is essentially embodied in the four programming steps of Claim 1.

As explained to the Examiner in the interview, and as argued in Applicant's previous response, McLaughlin teaches a method of configuring I/O that interfaces with the programmable logic controller, but does not teach a graphical user interface for programming a PLC to direct a process as recited in the claims. Indeed, McLaughlin recognizes that the main advantage and purpose of the method of McLaughlin is that "the programmer only needs to program the logic of the control algorithms [i.e., the control process] and does not need to program where the various I/O devices will be located or when the various I/O devices were installed in the generic control program. In other words, the Virtual Rack module 500 resembles a configuration language, rather than a process control system 100." (Column 7, Lines 52-59).

Most importantly, McLaughlin provides nothing, other than conventional ladder logic programming methods, for performing the actual programming of the PLC operating program, i.e., program 2602. Specifically, and referring to FIG. 2 of McLaughlin, McLaughlin provides that the ladder logic program 204 is developed on programming terminal 108 (i.e., a PC) and downloaded to memory 202 of the PLC 102. McLaughlin teaches that standard programming software provided by the PLC manufacturer (e.g., "PLC-5 Programming software (6200 Series or AI series from Allen-Bradley Company")) is used to program the PLC to direct a process. (Column 5, Lines 13-16). McLaughlin **does not** provide for a new method of writing the control program 204 of McLaughlin, but rather relies on, as an example, conventional ladder logic programming. This standard programming software should be the emphasis of analysis, not the configuration capabilities of McLaughlin. Applicant submits that it has consistently sought to differentiate its claims from this conventional programming methodology.

To that end, and per what Applicant believes was suggested by the Examiner, Claims 1, 9, 17 and 20 have been amended to emphasize the use of a graphical user interface as claimed in the programming of the PLC to direct a process. As explained above and discussed in the interview, programming a PLC to direct a process is not configuring the I/O with which the PLC

interacts. The preamble has been amended to recite "A method of programming a programmable logic controller to direct a process." The preamble, prior to amendment, already recited that the programmable logic controller directs a process through output signals at the outputs in response to input signal at the inputs, so this amendment emphasizes that it is this previously recited "process" that is being programmed.

The displaying step has also been amended to emphasize that the PLC directs a process via sequential steps that need to be programmed. Specifically, the displaying step has been amended as follows: "displaying to a user on a monitor a graphical data entry user interface for a plurality of sequential steps to be directed by said PLC." This claim element also recites (and did recite prior to amendment) the type of graphical data entry user interface that is displayed, i.e., "said graphical data entry user interface representing respective inputs to be monitored by said programmable logic controller at each of said sequential steps and respective outputs to be initiated by said programmable logic controller at respective ones of said sequential steps." So, in summary, the interface shows a plurality of steps "to be directed by said PLC" where each of these steps displayed in the graphical data entry user interfaces is shown with possible inputs to be monitored at that step and outputs to be initiated by that step. This display step is best understood as shown in FIG. 3 of the application, where the graphical interfaces shows 32 possible sequential steps (shown as STEPS 0 to 31), along with 15 possible inputs to be monitored at each step (Inputs 0 to 14) and 15 possible outputs to be triggered at each step (shown as Outputs 0 to 15).

Claim 1 then affirmatively recites that the recited graphical data entry user interface is used to program the steps the PLC uses to direct the process. Specifically, Claim 1 recites "receiving for said process" (i.e., the process that the PLC is being programmed to direct) "via said graphical user interface, an identification of at least one input selected by said user to be monitored for at least one of said sequential steps" (i.e., the sequential steps displayed in the interface) "and an identification of at least one output selected by said user to be initiated for said at least one sequential step" (i.e., if the user programs with the interface that inputs should be monitored for a given step, the user also programs outputs to be triggered for that step). With

reference to the embodiment of FIG. 3, the user programs each step by adding simple check marks in the boxes on the check grid under the desired inputs and outputs.

Certainly, this recited methodology is distinguishable from the virtual I/O configuration taught by McLaughlin. Indeed, as explained at length in the previous response, configuring I/O (i.e., address locations of I/O, model type, voltage needs, etc.) involves making sure that the PLC knows how to interact with individual I/O when the time is necessary, but does not tell the PLC what steps to take in order to direct a process. By way of analogy, configuring I/O is like giving an employee a phone book with people's phone numbers. Programming the PLC is like giving the employees an instruction manual telling the PLC when and under what circumstances to call specific people from the phone book.

The claimed methodology is also vastly different from conventional methods of programming a PLC to direct a process, such as the software that is sold with the PLC. In order to use such software, a programmer needs to be versed in a programming language such as ladder logic. That person needs to know different programming commands which the user types out or selects in sequential order in order to program the process. Even if the Examiner were to somehow construe the interface between the user and this prior art software to be a graphical user interface, it would not be the graphical interface that Applicant claims as being displayed to the user, i.e., one that displays a plurality of sequential steps to be directed by the PLC, where the interface represents for each displayed sequential step the inputs to be monitored and outputs to be initiated.

The Prior Art programming methods essentially start with a blank slate and require the author to prepare the program, process step by process step. This prior art methodology does not display a configurable graphical data entry interface that already displays a plurality of possible steps that merely require the entry or selection of inputs and outputs for each step in order to develop a control program. An example of such a Prior Art programming method is the DOS based 6200 Series PLC-5 Programming Software from Allen Bradley referenced in the Specification of McLaughlin, Column 5, Lines 14-16. Being a DOS based programming

method, the programmer is required to laboriously type out an entire program, line by line and command by command.

Later versions of ladder logic programming methods available from PLC manufacturers adapted this DOS-based ladder logic programming tool to the WINDOWS environment, but still suffered from the shortcomings of the prior art, namely that the programmer needed to start from a blank slate of sorts to program the sequential steps of the process to be directed by the PLC, the programmer had to enter individual instructions for each step, and the programmer needed to type out and enter each individual input and output for each step. To illustrate the difference between the claimed programming method and this ladder logic programming method, the inventor has created a document (attached as Exhibit A) entitled "PROJECT START USING ALLEN BRADLEY RSLOGIX(TM) LADDER LOGIC PROGRAMMING SOFTWARE." This document, via textual description from the inventor and screen shots, clearly illustrates the complexity of programming **even a very simple two-step PLC program** to turn on a light. Comparing this document to the graphical interface described in the present application and claimed herein evidences the vast improvement the Applicant has achieved over the prior art programming methodologies.

In summary, (1) Applicant maintains that McLaughlin does not teach anything more with respect to programming a programmable logic controller to direct a process than conventional ladder logic programming techniques; (2) Applicant's claims recite steps in using a specific graphical user interface for programming the sequential steps used by the PLC to direct a process; (3) the specified graphical user interface recited in the claims, and the specific steps of using the graphical data entry user interface, are neither taught nor suggested by the prior art of record; and (4) Applicant's amendments do not necessitate a new search or further consideration by the Examiner.

Therefore, per the foregoing, it is submitted that Independent Claims 1, 9, 17 and 20 are not anticipated by the art of record, including McLaughlin.

It is further submitted that dependent Claim 2-8, 10-16, 18-19 and 21-22, which depend from the allowable independent claim, are also allowable. Briefly though, Claims 6 and 14 expressly recite that the claimed graphical data entry user interface is a check grid. The Examiner takes "[o]fficial Notice . . . that the use of check grids are well known in the art." This conclusion is respectfully traversed by the Applicant. MPEP 2144.03(A) states that "Official notice unsupported by documentary evidence should only be taken by the examiner where the facts asserted to be well known, or to be common knowledge in the art are capable of instant and unquestionable demonstration as being well-known." Applicant is well versed in the art of programming programmable logic controllers. It is submitted that, notwithstanding the fact that the prior art of record does not teach a programming method using a graphical user interface as recited in the independent claims, the use of check grids in programming PLCs to direct a process is not well known in the art so as to be subject to Official Notice, at least based on Applicant's extensive experience in programming PLCs. Documentary evidence of the use of check grids in programming PLCs to direct a process is respectfully requested.

B. Claim Rejection under 35 U.S.C. § 103

Applicant submits that the obviousness rejection of Claims 2-4, 6-7, 10-15, 18, 19, 21 and 22 has been addressed above. For at least the reasons set forth above, these claims are allowable over the cited references.

C. New Claims

New Claims 49-74 have been added, examination of which are respectfully requested.

Claim 49 is also directed to a method of programming a programmable logic controller to direct a process. A graphical data entry user interface is displayed to a user representing a plurality of programmable sequential steps for programming by said user to be executed by said programmable logic controller in directing a process. For each step of a control program being programmed by said user via said graphical data entry user interface requiring monitoring of inputs and/or initiation of outputs coupled to said programmable logic controller, an

identification of any inputs selected by said user to be monitored and an identification of any outputs selected by said user to be initiated are received, where each step of said control program being programmed by said user corresponds to a respective step from said plurality of programmable sequential steps from said graphical data entry user interface. The identifications of said inputs and outputs selected by said user are converted into data elements to be provided to said programmable logic controller for execution as part of said control program.

Claim 49 is directed to the embodiment where the graphical data entry user interface represents a plurality of programmable sequential steps and steps from a control program being programmed by a user are programmed by configuring steps from the plurality of programmable sequential steps of the graphical data entry user interface.

Claims 50-57 depend from Claim 49. Claims 58-72 parallel claims from Claims 50-57 and are directed to programming apparatus and computer readable medium.

Claim 73 is similar to Claim 49 and further directed to the embodiment where the selectable inputs and outputs of the programmable sequential steps are represented by selectable graphical identifiers. Claim 73 is also directed to the embodiment where the programmable sequential steps also have a timer enable command and a timer value is received for enabled timer steps.

IV. Conclusion

In view of the foregoing remarks and amendments, Applicant submits that this application is in condition for allowance at an early date, which action is earnestly solicited.

The Commissioner for Patents is hereby authorized to charge any additional fees or credit any excess payment that may be associated with this communication to deposit account **04-1769**.

Respectfully submitted,

Dated: 6/2/05

Joseph A. Powers, Reg. No.: 47,006
Attorney For Applicant

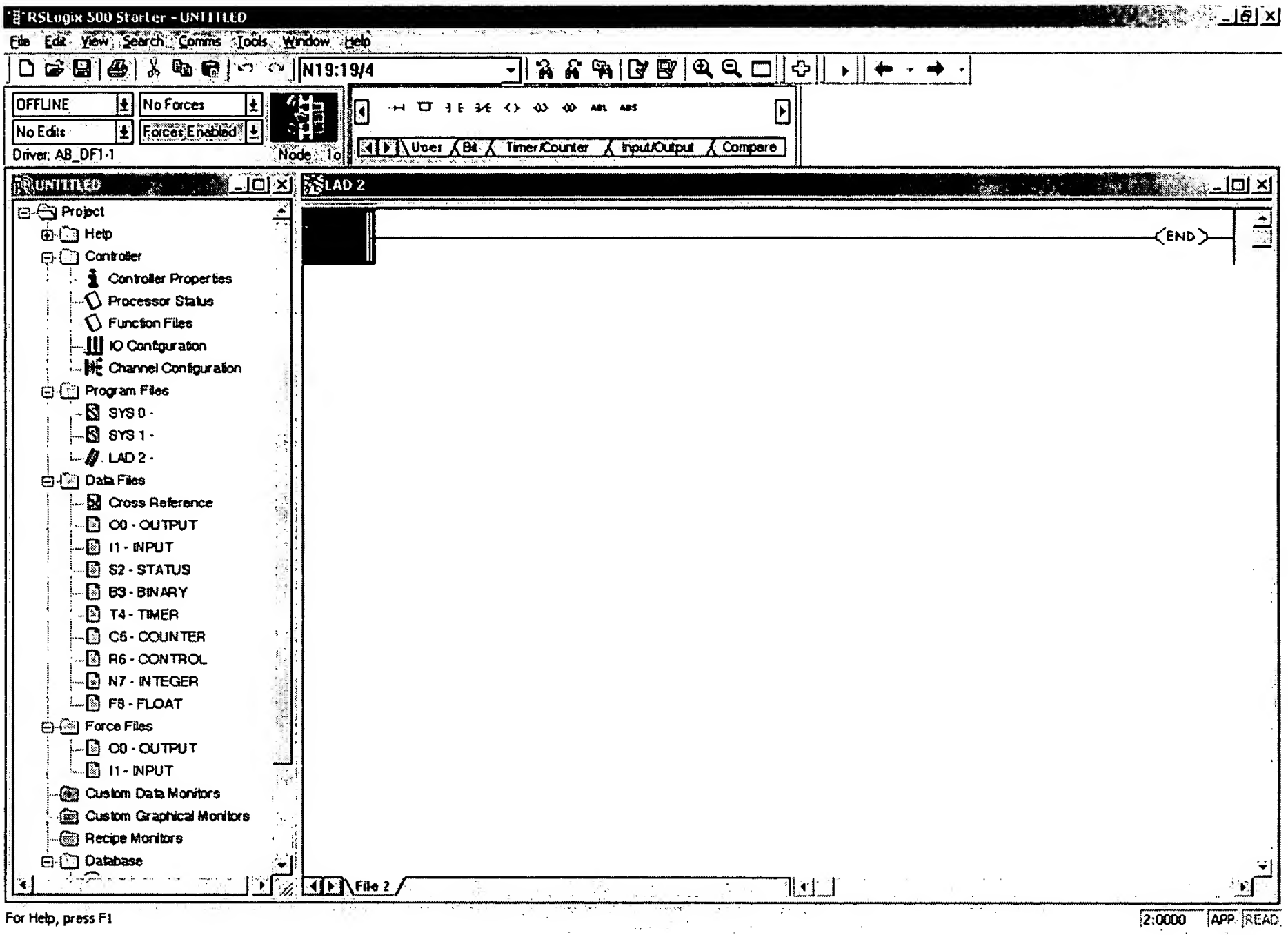
DUANE MORRIS LLP
One Liberty Place
Philadelphia, Pennsylvania 19103-7396
(215) 979-1842 (Telephone)
(215) 979-1020 (Fax)

EXHIBIT A

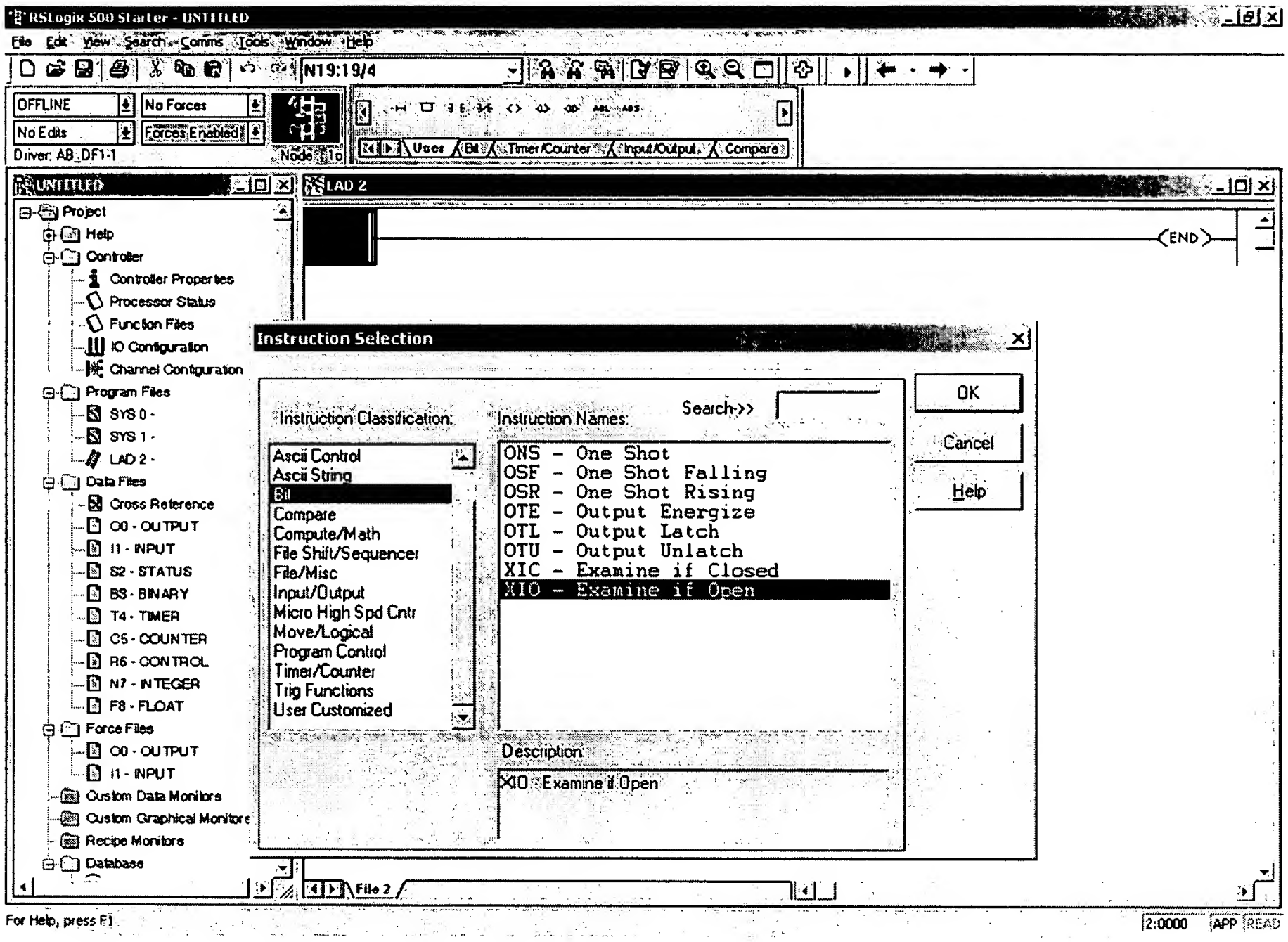
To Response and Amendment filed in Response to Final Official Action Dated April 6, 2005 in U.S. Application Serial No. 09/772,493

PROJECT START USING ALLEN BRADLEY RSLOGIX (TM) LADDER LOGIC PROGRAMMING SOFTWARE.

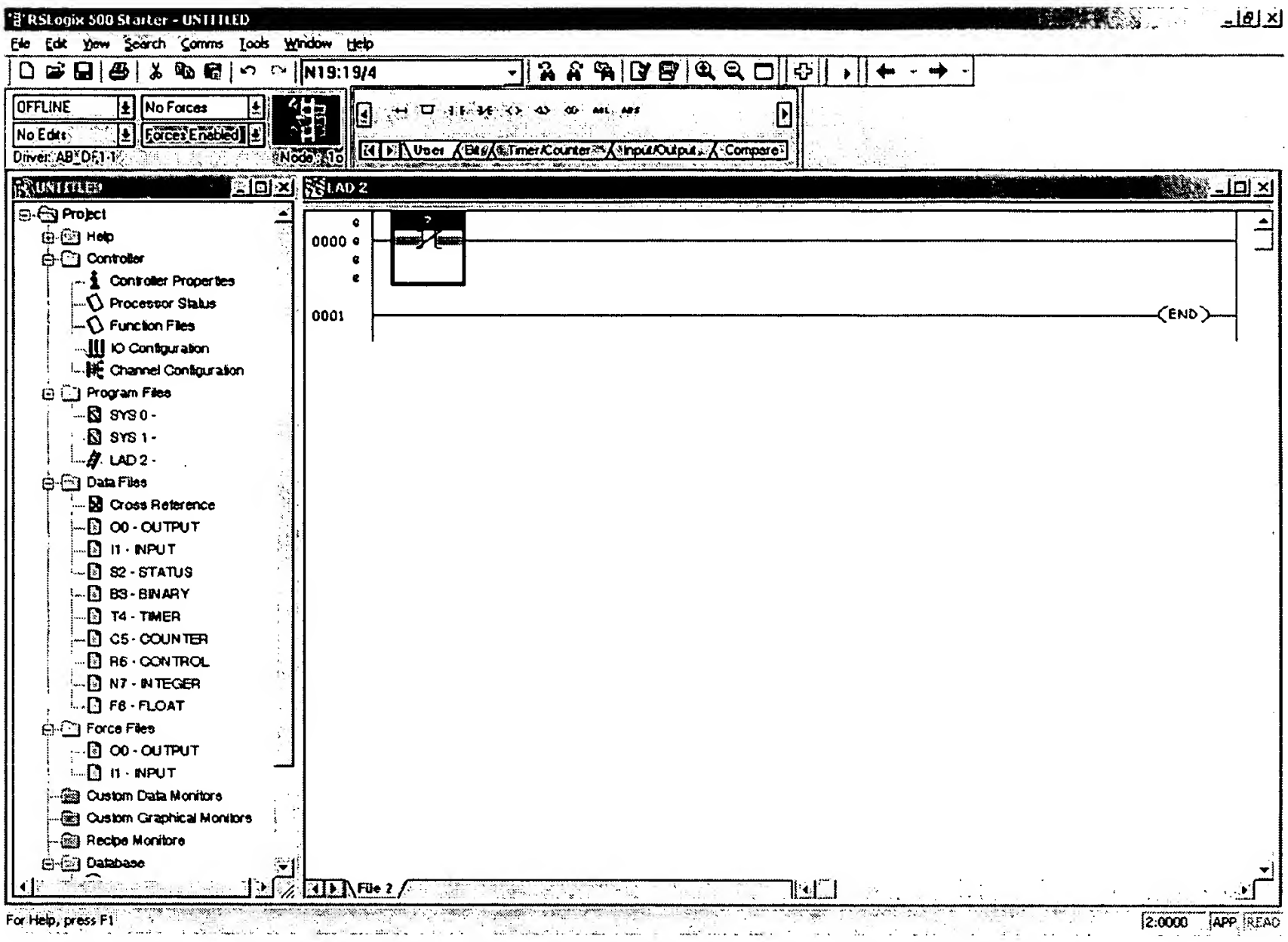
STEP 1: Start a new project using the software. The following blank project screen will appear. The programmer must now design the logic, and determine which instructions will be used.



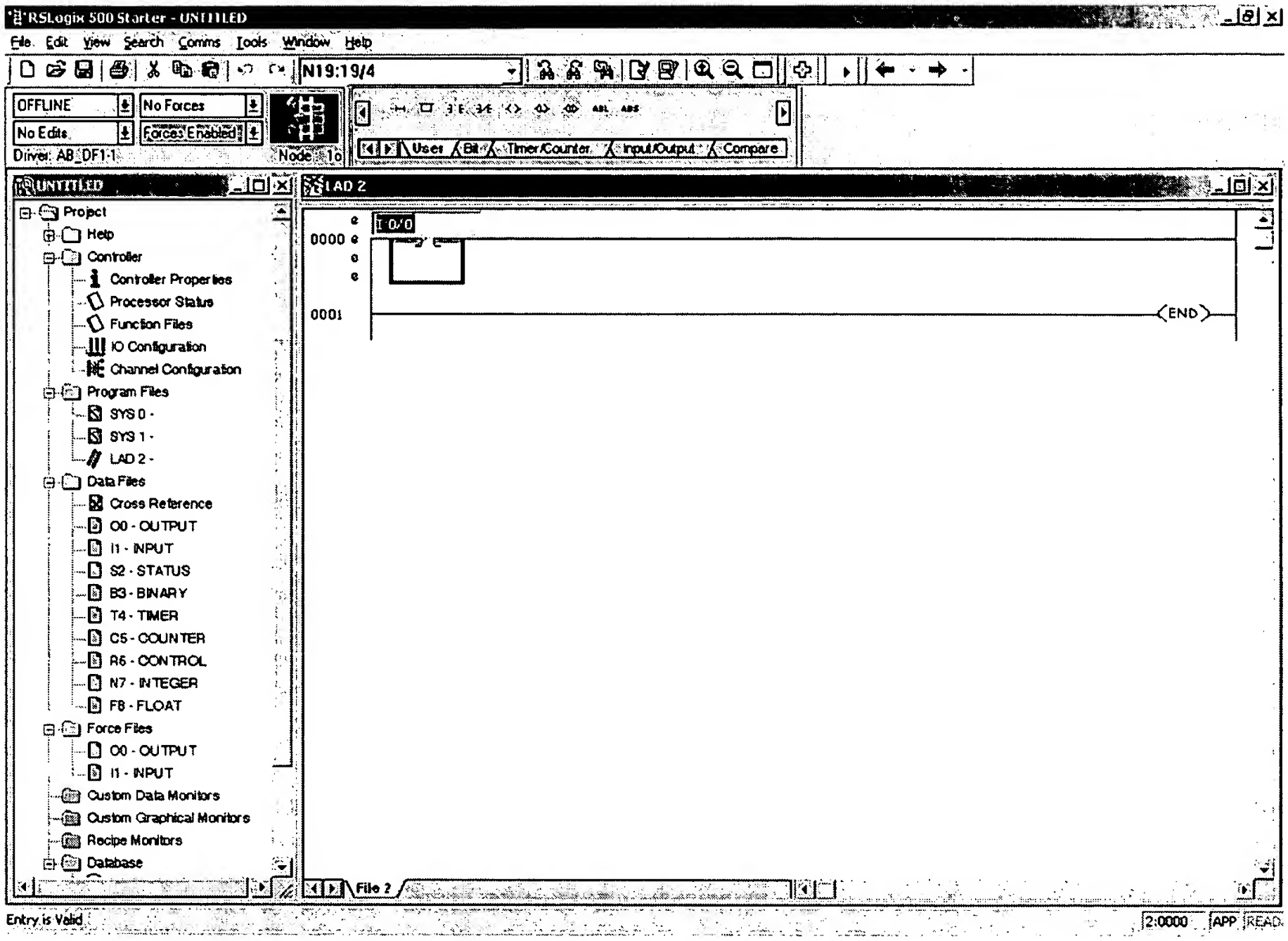
STEP 2: The programmer goes into instruction selection mode. The following menu pops up listing the hundreds of different available instructions. The programmer must know the function of each of these instructions and pick the appropriate one for the application. Some basic instructions are available from a toolbar at the top middle of the screen. In this example, we are entering in a XIO or normally open contact.



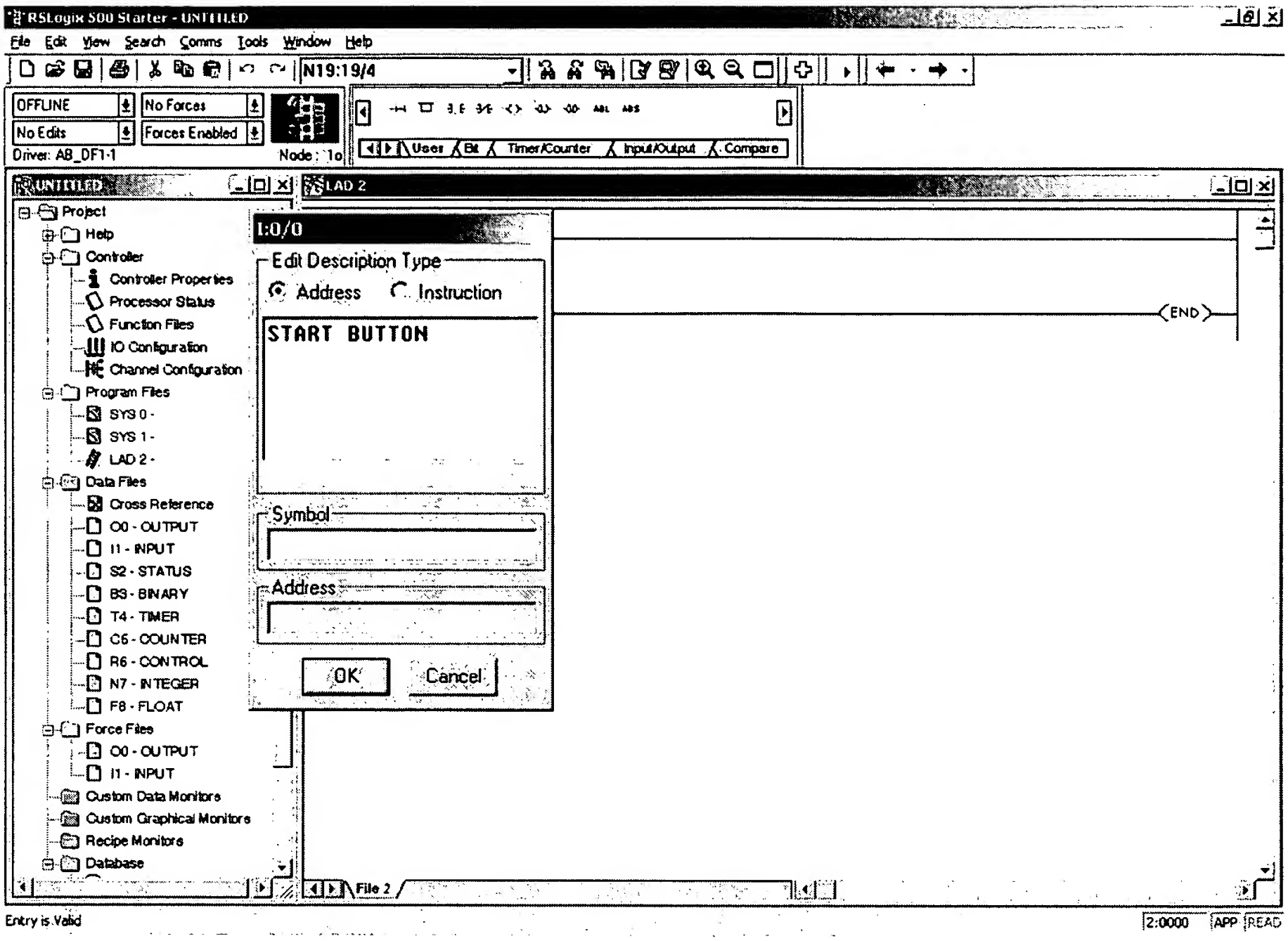
STEP 3: The programmer enters the instruction, and the software shows the contact in the ladder rung as shown below.



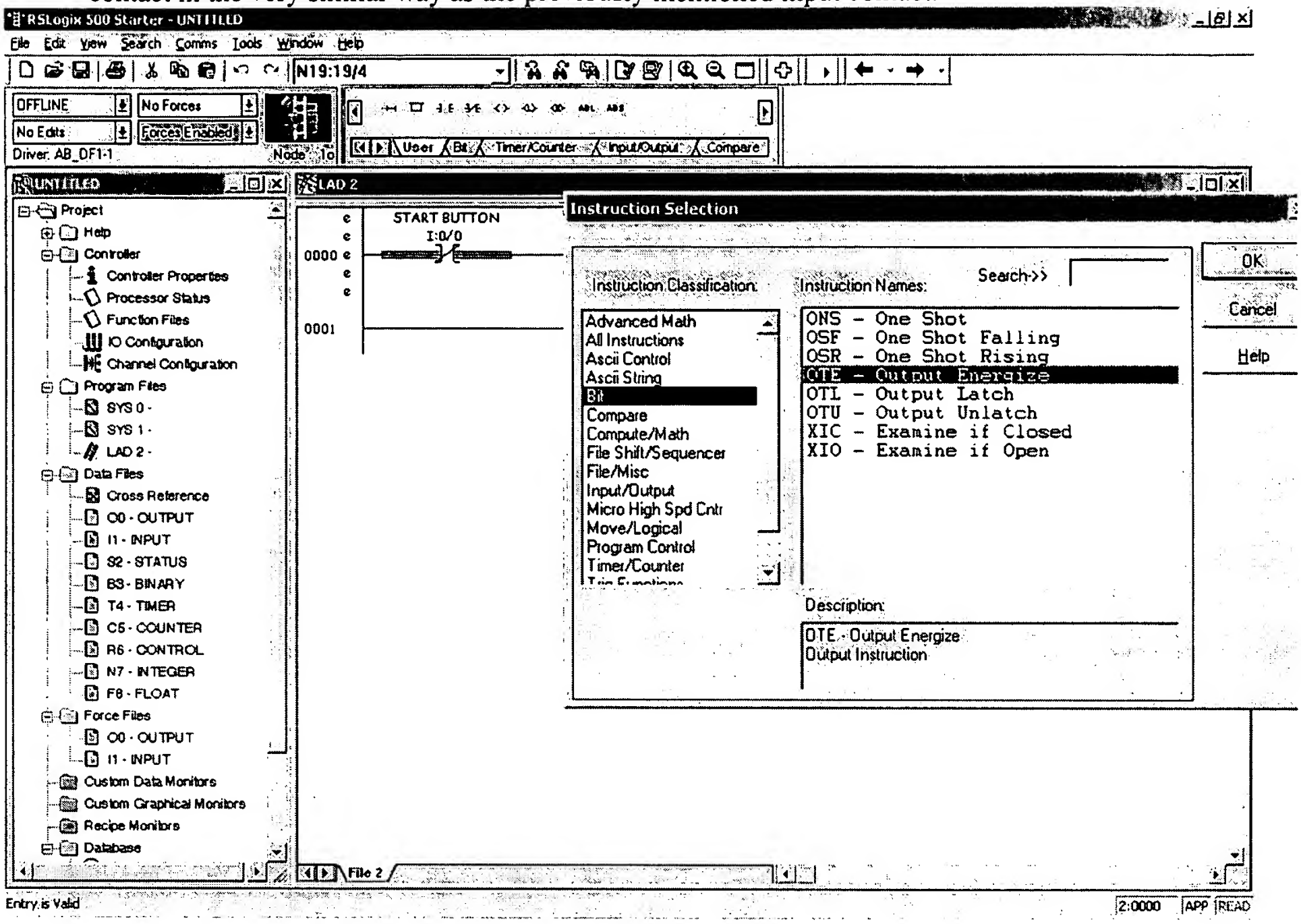
STEP 4: The input contact must now be addressed. This is done by the programmer to let the program know what input should be used in this ladder. The programmer must know the correct syntax, and also the proper physical location of this input. In this example, we are addressing the input contact as I: 0/0 by textual entry.



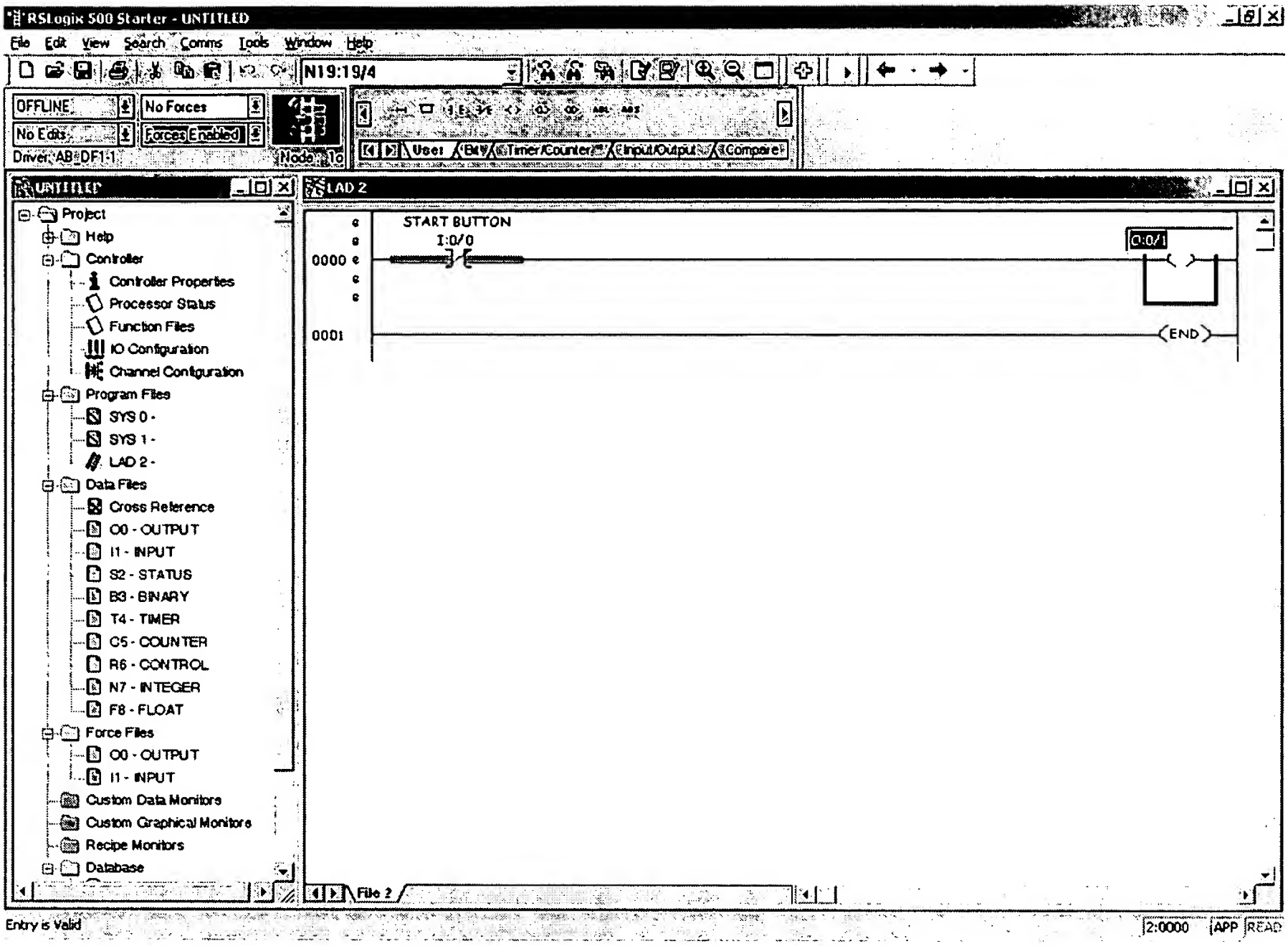
STEP 5: A comment is now entered for this input contact. This is used for documentation purposes. A separate window pops up for this task. In this example, the programmer has labeled this address, START BUTTON



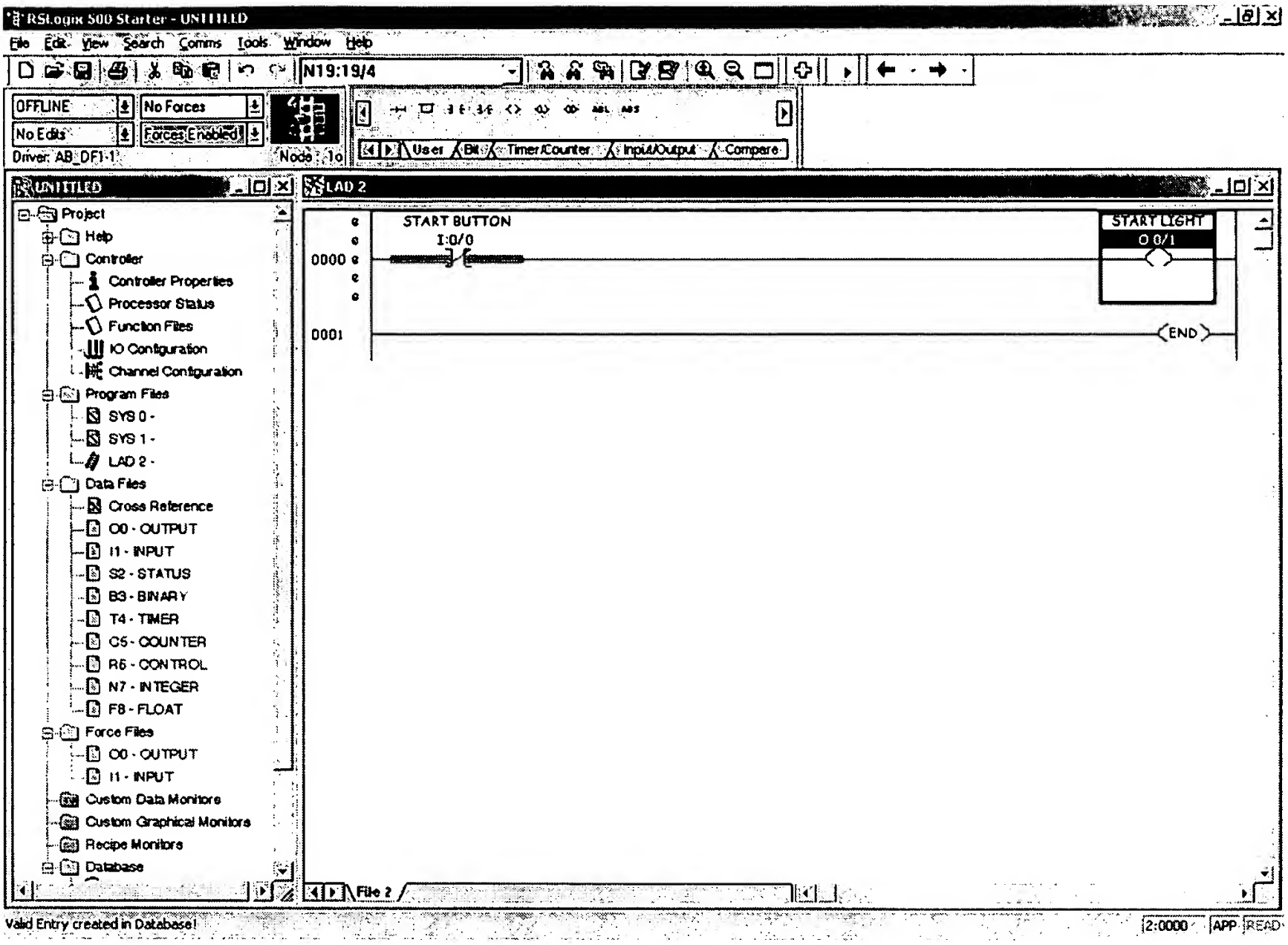
STEP 6: The next task is to put an output contact at the end of the ladder rung. All ladder logic programming must strictly follow this format. The programmer selects an output contact in the very similar way as the previously mentioned input contact.



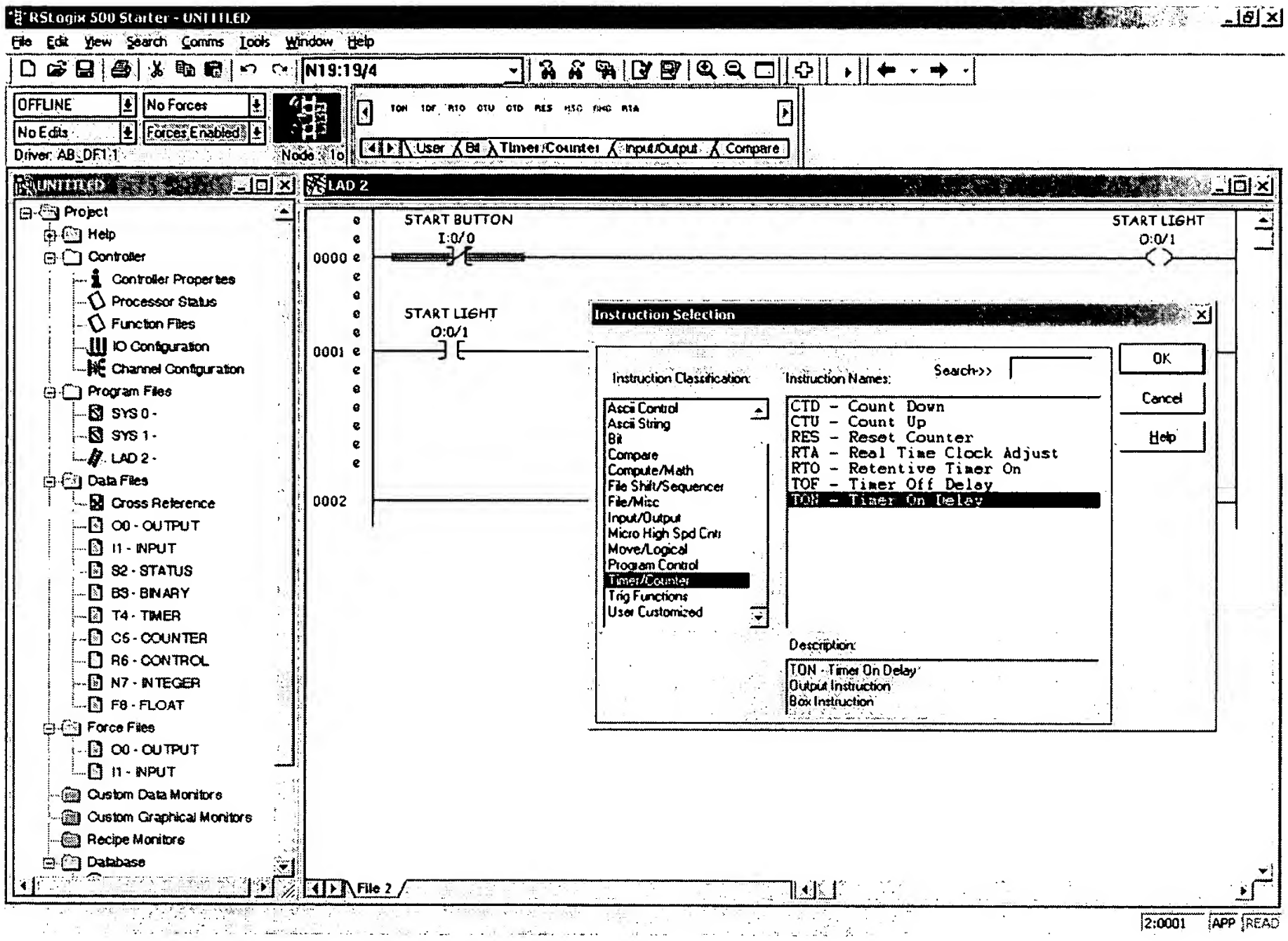
STEP 7: After the contact has been selected, it must be addressed and commented by textual entry.



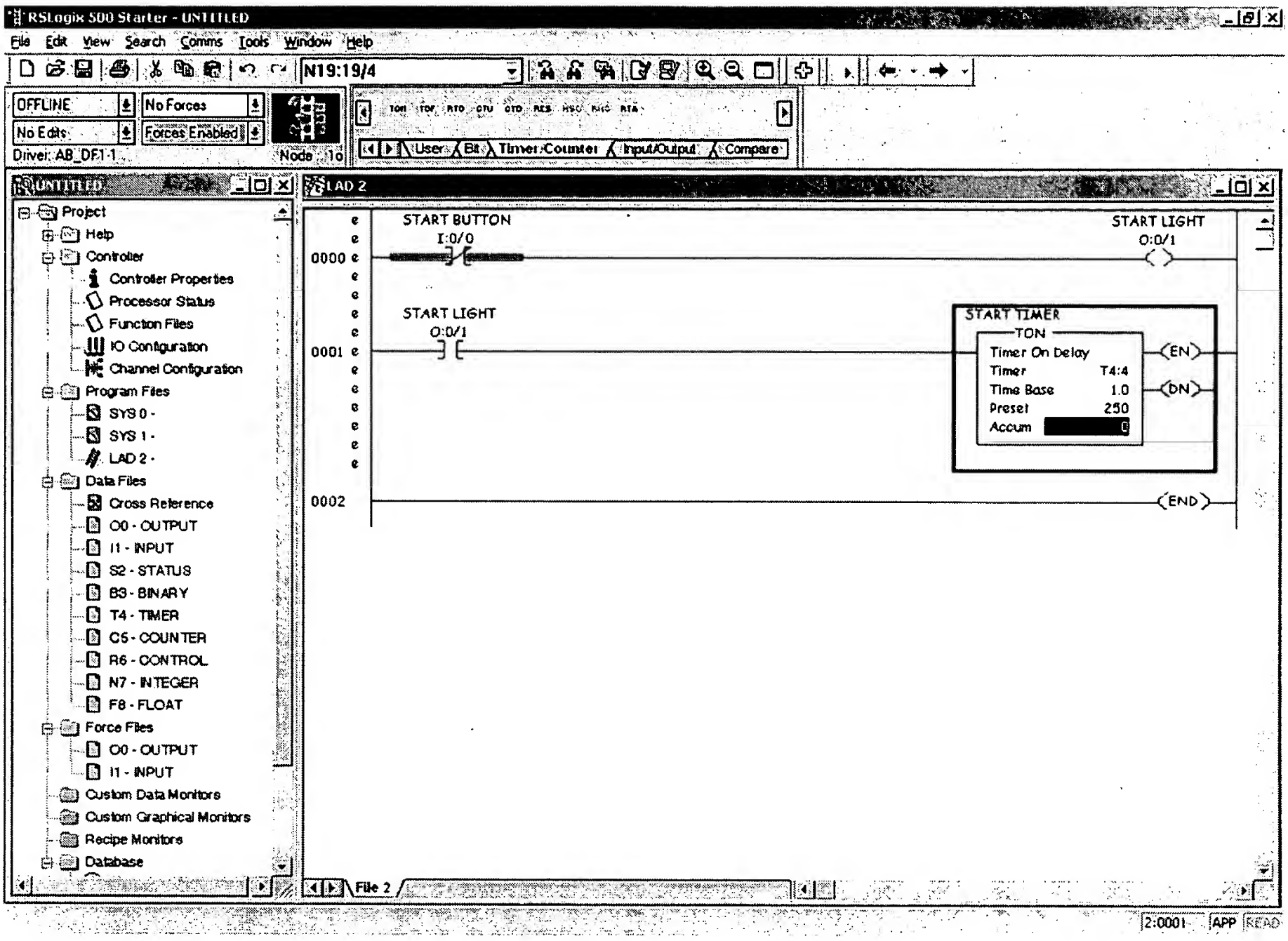
STEP 8: The first ladder logic rung is now complete. The completed rung is shown below.



STEP 9: Timer Insert. In this step, the programmer needs to add a timer that activates after the output comes on. They must insert a new rung, and insert the contact onto that rung in the same way as described above. The programmer must now select and insert the proper timer.



STEP 10: The timer has been selected, and the programmer now must enter in the timer address, time base, and also timer preset. The timer output must then be used in another rung to activate physical devices off of this timer. The completed rung is shown below.



STEP 11: The program must now be compiled and downloaded to the PLC. During the compile phase, errors and incorrect syntax are checked. Any errors will stop the process, and the programmer must search the code for errors.